A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem

Rubén Ruiz[1*], Thomas Stützle[2]

[1] Universidad Politécnica de Valencia. Departamento de Estadística e Investigación Operativa Aplicadas y Calidad. Grupo de Investigación Operativa.

Camino de Vera S/N, 46021 Valencia (Spain)

email: rruiz@eio.upv.es

[2] Darmstadt University of Technology. Department of Computer Science

Intellectics Group. Hochschulstr. 10, D-64283 Darmstadt (Germany)

email: stuetzle@informatik.tu-darmstadt.de

5th January 2005

Abstract

Over the last decade many metaheuristics have been applied to the flowshop scheduling problem, ranging from Simulated Annealing or Tabu Search to complex hybrid techniques. Some of these methods provide excellent effectiveness and efficiency at the expense of being utterly complicated. In fact, several published methods require substantial implementation efforts, exploit problem specific speed-up techniques that cannot be applied to slight variations of the original problem, and often re-implementations of these methods by other researchers produce results that are quite different from the original ones. In this work we present a new iterated greedy algorithm that applies two phases iteratively, named *destruction*, were some jobs are eliminated from the incumbent solution, and *construction*, where the eliminated jobs are reinserted into the sequence using the well known NEH construction

*Corresponding author

heuristic. Optionally, a local search can be applied after the construction phase. Our iterated greedy algorithm is both very simple to implement and, as shown by experimental results, highly effective when compared to state-of-the-art methods.

Keywords: Iterated Greedy, Flowshop Scheduling, Metaheuristics.

1 Introduction

One of the most thoroughly studied scheduling problems is the flowshop problem (FSP). In the FSP, a set $N = \{1, \ldots, n\}$ of n independent jobs has to be processed on a set $M = \{1, \ldots, m\}$ of m machines. Every job j, $j \in N$, requires a given fixed, non-negative processing time p_{ij} on every machine i, $i \in M$. In a flowshop, all n jobs are to be processed on the m machines in the same order, that is, the jobs follow the same flow in the shop starting from machine 1 and finishing on machine m. The objective is to find a sequence for processing the jobs in the shop so that a given criterion is optimized. The criterion that is most commonly studied in the literature is the minimization of the total completion time, also called makespan (C_{max}), of the production sequence. A common simplification in the flowshop scheduling problem is to avoid *job passing* in the sequence; that is, the processing sequence on the first machine is maintained throughout the remaining machines. The resulting problem is called the permutation flowshop problem (PFSP); with the makespan criterion it is denoted as $F/prmu/C_{max}$ (Pinedo, 2002). There are $n!$ possible sequences and the problem is known to be $\mathcal{NP}$-complete (Rinnooy Kan, 1976).

The PFSP was first studied by Johnson in 1954 and since then, many heuristic and metaheuristic methods for its solution have been proposed. The methods range from the simple but effective *Rapid Access* (RA) heuristic of Dannenbring (1977) and the well known NEH heuristic of Nawaz et al. (1983) to the latest advanced metaheuristics like the ant-colony algorithm of Chandrasekharan and Ziegler (2004). An up-to-date review of flowshop heuristics and metaheuristics can be found in Ruiz and Maroto (2004a). Several recent metaheuristic techniques do provide excellent results in acceptable computation times but one possible criticism might be the excessive complexity of some of the proposed methods. In some cases, the algorithms published are so intricate that an independent coding is unlikely to obtain the same effectiveness or efficiency. Furthermore, some algorithms strongly exploit PFSP specific features that makes their adaptation to other flowshop variants very difficult or even impossible. In this paper, we propose a new iterated greedy (IG) algorithm for the PFSP that is based on the NEH heuristic. The IG algorithm we present is very simple to code and easily adaptable to other flowshop problems. In addition, it is very effective as shown by the fact that we found new best solutions for three PFSP instances. This latter fact is especially remarkable because of the high level of sophistication reached by the algorithmic techniques that have been applied to the PFSP. However, in our opinion the most important advantage of our IG algorithm is its conceptual simplicity, which makes it easily tunable (our algorithm only has two parameters) and extendible to other problems.

This paper is organized as follows. In Section 2, we provide a brief review of the most important and recent developments in the flowshop scheduling literature. Section 3 gives a detailed description of the iterated greedy algorithm and the procedure we used for parameter tuning. In Section 4 we present a complete comparative evaluation of the effectiveness and efficiency of the proposed IG algorithm as well as a comparison to some of the best known and most recent heuristics and metaheuristic approaches. We end with some concluding remarks in Section 5.

2 Literature review

The PFSP has been a very active research area on heuristic and metaheuristic techniques, mainly because of the difficulty encountered by exact methods to solve medium or large instances.

In the pioneering work of Johnson (1954), the author proposed a simple rule to obtain optimal sequences for the PFSP with two machines. This work raised significant interest in the PFSP and was followed by several attempts for solving the PFSP with more than two machines. Subsequent work includes the one on the CDS heuristic by Campbell et al. (1970) and constructive and improvement heuristics by Dannenbring (1977). The NEH heuristic by Nawaz et al. (1983) is regarded as the best performing heuristic method (see Turner and Booth, 1987 or Ruiz and Maroto, 2004a). There is no clear evidence to support the idea that substantially better heuristic algorithms have appeared since then. However, there are a number of several recent, interesting proposals, like the improvement heuristic of Suliman (2000) or the study of the NEH heuristic by Framinan et al. (2003), which also considers several extensions of NEH when facing other objectives than C_{max}.

Metaheuristic algorithms for the PFSP appeared much later than the heuristic counterparts.

Among the earliest approaches we find the simulated annealing of Osman and Potts (1989), which is fairly simple in the sense that it uses a constant temperature and an insertion neighborhood along with a first come, first served dispatching rule initialization procedure. The tabu search algorithm of Widmer and Hertz (1989), known as SPIRIT, consists of two phases where in the first one an initial solution based on the Open Traveling Salesman Problem (OTSP) is obtained. The second phase is a straightforward tabu search that works with the exchange neighborhood. Other noteworthy tabu search algorithms were proposed by Taillard (1990) and Reeves (1993). These two algorithms worked with the insertion neighborhood and used the NEH heuristic for obtaining an initial solution. Genetic algorithms for solving the PFSP have also appeared in Chen et al. (1995), Reeves (1995) or more recently in Wang and Zheng (2003) and Aldowaisan and Allahvedi (2003). These algorithms have in common that the initialization of the population is not random as in many other genetic algorithms. In this case, the population contains individuals obtained from the application of some of the aforementioned heuristics, including NEH, as well as random permutations. Other work includes the iterated local search algorithm by Stützle (1998) or the recent ant colony optimization algorithm of Chandrasekharan and Ziegler (2004). These two latter algorithms also rely on an NEH initialization.

All the aforementioned work has in common that the algorithms proposed are easy to code and therefore the results can be reproduced easily. In addition, many of the above algorithms are easily adaptable to other flowshop environments like flowshops with sequence dependent setup times or even complex hybrid flowshops (see Ruiz et al., 2004a or Ruiz and Maroto, 2004b). Additionally, there exists a number of powerful methods, usually in the form of highly elaborated single or hybrid techniques. Examples are the path-based method of Werner (1993), the well known tabu search algorithm of Nowicki and Smutnicki (1996), the genetic algorithm with path relinking of Reeves and Yamada (1998) or some hybrid techniques like the genetic algorithm with local search or simulated annealing of Murata et al. (1996), the genetic algorithm with local search of Ruiz et al. (2004b), the tabu search and simulated annealing of Moccellin and dos Santos (2000), the parallel simulated annealing algorithm of Wodecki and Bożejko (2002) or the very recent tabu search algorithm of Grabowski and Wodecki (2004). Some of these algorithms do provide excellent results and constitute some of the state-of-the-art methods available. However, they do have an important drawback: they are very sophisticated and an arduous coding task is necessary for their implementation. Furthermore, due to the often limited descriptions given in the original papers, it is often not possible to obtain a working algorithm showing similar performance unless the authors are contacted for details or the source code is provided. Additionally, the results given in the original work are obtained after many tweaks and speed-ups that are often not explained thoroughly to the reader. Many of these methods exploit PFSP specific features that are rarely applicable to extensions of the PFSP; for example, the critical path concept used in Nowicki and Smutnicki (1996), Reeves and Yamada (1998) or Grabowski and Wodecki (2004) does not hold when other situations like sequence dependent setup times or other objectives are considered in flowshops.

In this paper we aim at providing a simple and easily adaptable, yet powerful algorithm that tries to overcome these aforementioned problems.

3 Iterated Greedy algorithm for the PFSP

In a nutshell, iterated greedy (IG) generates a sequence of solutions by iterating over greedy constructive heuristics using two main phases: destruction and construction. During the destruction phase some solution components are removed from a previously constructed complete candidate solution. The construction procedure then applies a greedy constructive heuristic to reconstruct a complete candidate solution. Once a candidate solution has been completed, an acceptance criterion decides whether the newly constructed solution will replace the incumbent solution. IG iterates over these steps until some stopping criterion is met. IG is closely related to Iterated Local Search (ILS): instead of iterating over a local search as done in ILS (Lourenço et al. 2002), IG iterates in an analogous way over greedy construction heuristics.

IG has been applied with success, for example, to the Set Covering Problem (SCP) by Jacobs and Brusco (1995) and by Marchiori and Steenbeek (2000). However, so far IG has not been applied to scheduling problems. For the application of IG to the PFSP we use as the greedy construction heuristic the well known NEH method of Nawaz et al. (1983) which can be described by the following three steps:

1. For each job compute the total processing time on the m machines:

$$\forall j, j \in N, P_j = \sum_{i=1}^{m} p_{ij}$$

2. Sort the jobs in descending order of P_j. Let the resulting sequence be ϕ. Next, the first two

jobs (that is, $\phi(1)$ and $\phi(2)$—the ones with the largest total processing time) are chosen and the two possible sequences of these two jobs are evaluated.

3. Finally, repeat the following steps until all jobs are sequenced. In the ith step, the job $\phi(i)$ at position i is taken and tentatively inserted into all the possible i positions of the sequence of the jobs that are already scheduled. Select of these the sequence that results in the minimum makespan. For example, if $i = 4$, the previously built sequence would contain the first three jobs of the list ϕ generated in step 2; then, the fourth job could be placed either in the first, the second, the third or the last position of the sequence. The best sequence of the resulting four sequences would be selected for the next iteration.

NEH evaluates a total of $[n(n + 1)/2] - 1$ schedules; n of these schedules are complete sequences. This makes the complexity of NEH rise to $\mathcal{O}(n^3m)$ which can lead to considerable computation times for large problem instances. However, Taillard (1990) introduced a data structure that allows to reduce the complexity of NEH to $\mathcal{O}(n^2m)$. In this data structure, the makespan of all n possible insertion points can be calculated by means of three matrices, named e (heads), q (tails) and f (heads plus processing times). As a matter of fact, one does not need to explicitly calculate the makespan with these matrices. In our implementation we use this data structure, since it is rather straightforward to implement (see Taillard, 1990 for more details).

The job sequence obtained by applying the NEH heuristic gives the initial solution of our IG algorithm. The NEH heuristic is also used in the construction phase of our IG algorithm.

3.1 Destruction and Construction phases

Two central procedures in any IG algorithm are the destruction and the construction procedures. The destruction procedure is applied to a permutation π of n jobs and it chooses randomly, without repetition *d* jobs. These *d* jobs are then removed from π in the order in which they were chosen. The result of this procedure are two subsequences, the first being the partial sequence π_D with $n - d$ jobs, that is the sequence after the removal of d jobs, and the second being a sequence of *d* jobs, which we denote as π_R. π_R contains the jobs that have to be reinserted into π_D to yield a complete candidate solution in the order in which they were removed from π.

The construction phase consists in the application of step 3 of the NEH heuristic until a complete sequence of all n jobs is obtained. The construction phase starts with subsequence π_D and performs *d* steps in which the jobs n π_R are reinserted into π_D. Hence, we start with π_D and insert the first job of π_R, $\pi_R(1)$, into all possible $n - d + 1$ positions of π_D. The best position for $\pi_R(1)$ in the augmented π_D sequence is the one that yields the smallest C_{max}. This process is iterated until π_R is empty.

3.2 Local Search and acceptance criterion

It is also straightforward to add a local search procedure to IG; this can be done by improving each solution that is generated in the construction phase with a local search. There are many different alternatives for a local search algorithm that can be considered. Following the idea of having a simple and easily implementable algorithm, we choose a rather straightforward local search algorithm.

Our local search algorithm is based on the insertion neighborhood, which is commonly regarded as being a very good choice for the PFSP (see Osman and Potts, 1989, Taillard, 1990 or Nowicki and Smutnicki, 1996). The insertion neighborhood of a permutation π of jobs is defined by considering all possible pairs of positions $j, k \in \{1, \ldots, n\}$ of π, $j \neq k$ where the job at position j is removed from π and inserted at position k. The sequence that results from such a move is $\pi' = (\pi(1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(k), \pi(j), \pi(k+1), \ldots, \pi(n))$ if $j < k$, or $\pi' = (\pi(1), \ldots, \pi(k-1), \pi(j), \pi(k), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(n))$ if $j > k$. The set of insertion moves I is defined as $I = \{(j,k) : j \neq k,\ 1 \leq j,k \leq n \wedge j \neq k-1,\ 1 \leq j \leq n, 2 \leq k \leq n\}$ and the insertion neighborhood of π is defined as $V(I, \pi) = \{\pi_v : v \in I\}$. The size of the insertion neighborhood is $(n-1)^2$. The usage of the very same data structures proposed by Taillard to speed-up the NEH heuristic allows to evaluate the whole neighborhood of one solution in $\mathcal{O}(n^2m)$. In addition, it would be possible to consider the critical path concept and a representative insertion neighborhood as in Nowicki and Smutnicki (1996) or Reeves and Yamada (1998). While the representative neighborhood is faster to evaluate, it adds a significant degree of complexity to the local search and thus we refrain from using these concepts here.

We implemented a first-improvement type local search algorithm that is outlined in Figure 1.

`[Insert Figure 1 about here]`

After the destruction and construction phases and the optional local search phase we have to consider whether the new sequence is accepted or not as the incumbent solution for the next iteration. One of the simplest *acceptance criteria* is to accept new sequences only if they provide a better C_{max} value. However, an IG metaheuristic implementing this acceptance criterion may lead to stagnation situations of the search due to insufficient diversification. Hence, we considered a simple simulated annealing-like acceptance criterion (similarly to the one in Osman and Potts, 1989 and Stützle, 1998) with a constant temperature. This constant temperature depends on the instance size and is computed following the suggestions of Osman and Potts (1989) as

$$\text{Temperature} = T \cdot \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}, \tag{1}$$

where T is a parameter that needs to be adjusted.

The complete Iterated Greedy metaheuristic with the aforementioned local search and acceptance criterion is summarized in Figure 2. Our procedure has only two parameters, namely T and d.

`[Insert Figure 2 about here]`

3.3 An example application of the IG metaheuristic

To illustrate the functioning of the IG method, we show the application of one IG iteration to instance `Car8` of Carlier (see Beasley, 2004). This instance has $n = m = 8$ and the processing times are shown in Table 1. The example iteration is illustrated in Figure 3 without considering the local search phase and using $d = 3$

`[Insert Table 1 about here]`

`[Insert Figure 3 about here]`

The example shows that starting from the solution given by the NEH heuristic ($C_{max} = 8564$) a new solution of $C_{max} = 8366$ is reached by removing jobs 5, 1 and 4 and reinserting them in the way described above. This new solution is accepted by the acceptance criterion since it is better than the starting solution. Furthermore, this new sequence is known to be an optimal solution for instance `Car8`.

3.4 Experimental parameter tuning of the algorithm

A major advantage of the proposed IG algorithm is that, different from many other metaheuristics, it has only two parameters, T and d. We have used the Design of Experiments (DOE) approach for the parameter tuning of the algorithm (see Montgomery, 2000). More precisely, we have carried out a full factorial design using the two parameters as factors at the following levels (values):

- T, 6 levels: $0.0, 0.1, \ldots, 0.5$
- d, 7 levels: $2, 3, \ldots, 8$

This yields a total of 42 different "treatments" to be tested, one for each combination of the levels of the two factors. For the experiments we use a set of randomly generated PFSP instances. These instances have been generated using the procedure given in Taillard (1993). More precisely, we have considered 68 different combinations of n and m, with $n \in \{20, 50, 80, \ldots, 440, 470, 500\}$ and $m \in \{5, 10, 15, 20\}$, and the processing times are distributed uniformly in the interval $[1, 99]$. For every combination of n and m, we have generated five instances, resulting in a total of 340 instances. The 340 instances are tackled by the 42 different "treatments" with a computation time limit fixed to $n \cdot (m/2) \cdot 20$ milliseconds. Setting the time limit in this way allows more computation time as the number of jobs or the number of machines increases. The experiments were carried out on a cluster of PC/AT computers with Athlon XP 1600+ processors (1400 MHz) and 512 MBytes of main memory.

Once all the trials were done, we transformed the data and used as the response variable of the experiment the following:

$$\text{Relative Percentage Deviation } (RPD) = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100 \tag{2}$$

Where $Some_{sol}$ is the solution obtained by a given algorithm alternative on a given instance and $Best_{sol}$ is the lowest makespan obtained in any experiment of the same instance. We carried out two different experiments, one for the IG method and another considering the IG method with the local search phase. In both cases the results were very similar and in what follows we comment on the results of the second experiment in which the local search phase is used.

The experiment was analyzed by means of a multi-factor Analysis of Variance (ANOVA) technique where n and m are also considered as non-controllable factors. To apply ANOVA, it is necessary to check the three main hypothesis of ANOVA which are normality, homogeneity of variance (or homoscedasticity) and independence of residuals. The residuals resulting from the experimental data were analyzed and all three hypothesis could be accepted. For example, the normality can be checked by a Quantile-Quantile plot of the residuals or by checking how the residuals fit into a theoretical normal distribution. Numerical tests are also possible, in this case we can use the χ^2-square or the Kolmogorov-Smirnov test for normality. The ANOVA results can be seen in Table 2.

`[Insert Table 2 about here]`

The analysis indicates that the factors that determine instance size, that is, n and m, are very significant. The magnitude of the F-Ratio is a clear indicator of significance, specially when the p-values are all zero among statistically significant effects. n and m are non-controllable factors so we focus on the other two remaining factors. The destruction factor results in statistically significant differences in the response variable. Surprisingly, the different levels of parameter T do not yield statistically significant differences. This suggests that the proposed IG algorithm is rather robust with respect to T. For the analysis of the best level for the destruction parameter we can focus on the means plot shown in Figure 4.

`[Insert Figure 4 about here]`

We can see that very low and very high destruction levels (number of jobs removed from the sequence) do result in statistically worse algorithms. (Recall that overlapping Least Significant Difference (LSD) intervals—here, at the 95% confidence level—mean statistically equivalent levels for the studied factor). From the figure it seems that a destruction of 4 jobs is most adequate, although it is not statistically significantly different from 3 or 5 at a 95% confidence level. The means plot for the parameter T can be seen in Figure 5.

`[Insert Figure 5 about here]`

As we commented, there is no clear statistically significant difference between the different levels for T. However, it seems that a setting of $T = 0.5$ does provide better results than a setting of 0.0 for which only improving solutions are accepted. Further experiments with higher levels of T showed no further improvements.

Considering the non-controllable factors n and m, we can also analyze the best combinations of the destruction and temperature factors for each of the 68 groups of instances. For example, by looking at Table 2, the interaction between n and destruction is very significant, which means that the number of jobs to be destructed does depend heavily on the number of jobs in the instance. Analogous results also hold for the interaction between m and destruction. Although this type of analysis would result in a more fine-tuned IG, this may also lead to an over-tuning that complicates the algorithm.

As a result from the experimental analysis, we set the parameters $d = 4$ and $T = 0.4$ (this last value was chosen at random between all the statistically similar ones, it could have been any other).

4 Experimental evaluation

In this section we provide a comprehensive experimental evaluation and comparison of the proposed IG algorithm with other powerful methods. For the comparisons we use the well known standard benchmark set of Taillard (1993) that is composed of 120 instances ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. This benchmark set contains some instances that have proved to be very difficult to solve. In this benchmark set there are 10 instances for each given size. The instances of one particular size are denoted as $n \times m$ in the tables containing the experimental results.

For our experiments we consider two variants of iterated greedy. The first variant does not include a local search phase and is denoted as IG_RS; the second variant uses local search and is denoted as IG_RS$_{LS}$. To compare the performance of the IG algorithms to known techniques from the literature, we reimplemented 12 classical or recent, well-performing algorithms. In particular, we implemented the NEH heuristic of Nawaz et al. (1983) with the improvements of Taillard (1990) (NEHT); the simulated annealing of Osman and Potts (1989) (SA_OP); the tabu search algorithm of Widmer and Hertz (1989) (SPIRIT); the (pure) genetic algorithms of Chen et al. (1995) (GA_CHEN) and Reeves (1995) (GA_REEV); the hybrid genetic algorithm with local search of Murata et al. (1996) (GA_MIT); two recent genetic algorithms of Ruiz et al. (2004b) (GA_RMA and HGA_RMA); the genetic algorithm of Aldowaisan and Allahvedi (2003) (GA_AA); the iterated local search of Stützle (1998) (ILS); and two recent ant algorithms of Chandrasekharan and Ziegler (2004) (M-MMAS and PACO). All 14 algorithms were implemented in Delphi 7.0 and run on the full set of Taillard's instances using as a stopping criterion a computation time limit of $n \cdot (m/2) \cdot 60$ milliseconds on a PC with an Athlon XP 1600+ processor (1400 MHz) with 512 MBytes of main memory. Hence, all the algorithms are run under the same conditions. For each algorithm, $R = 5$ independent runs were performed with the only exception being NEHT, for which only one single run was done since NEH is a deterministic algorithm except for random tie-breaking. As the performance criterion we measure for each instance the

$$\text{Average Relative Percentage Deviation } (\overline{RPD}) = \sum_{i=1}^{R} \left(\frac{Heu_{sol_i} - Best_{sol}}{Best_{sol}} \cdot 100\right)/R, \qquad (3)$$

where Heu_{sol_i} is the solution given by any of the R repetitions of the 14 considered algorithms and $Best_{sol}$ is the optimum solution or the lowest known upper bound for Taillard's instances as of April 2004 (these values can be consulted in Taillard, 2004). The results for all algorithms (averaged by instance size) are provided in Table 3.

[Insert Table 3 about here]

We first comment on the experimental results of the re-implemented algorithms, before comparing their performance to the two IG algorithms. The first noteworthy result is the excellent performance of the NEHT construction heuristic (this is the only algorithm we tested that is not a metaheuristic), which yields an $\overline{ARPD}$ across all instances of 3.35%. ($\overline{ARPD}$ is the average deviation from the best known or optimal solutions averaged across all 120 instances.) Additionally, NEHT is very quick; it takes, for example, only slightly more than 250 ms on the largest instances with 500 jobs and 20 machines. In fact, recent studies (Ruiz and Maroto, 2004a) have confirmed the superiority of NEHT over most recent constructive heuristics.

From the classical metaheuristics, SPIRIT and GA_CHEN obtain rather poor results, even worse on average than NEHT. Certainly, both would benefit greatly from an NEH initialization, but it is very questionable whether they could reach the performance of the other algorithms. Much better performance is obtained by SA_OP, although it initializes the search at a randomly generated solution. SA_OP is one of the algorithms that is easiest to implement and it is remarkable that it reaches a level of performance comparable to several genetic algorithms including GA_AA (a rather straightforward genetic algorithm that was originally designed for the no-wait flowshop problem) and GA_MIT, which is a fairly complex GA that is hybridized with local search. Among the other genetic algorithms, the by far best performance with an $\overline{ARPD}$ of 0.57% is obtained by HGA_RMA. However, this performance comes at the expense of some added complexity. HGA_RMA is a hybrid steady state GA that includes several population operators as well as a local search very similar to the one shown in Figure 1. HGA_RMA also performs much better than ILS; however, ILS has the advantage that it is a rather simple, easily implementable algorithm compared to HGA_RMA. Finally, let us consider the two recent ant algorithms M-MMAS and PACO. They both improve significantly over an earlier ant algorithm for the PFSP by Stützle (1998) and also show very good performance compared to the other algorithms. Between PACO and M-MMAS, PACO provides slightly better results for almost all instance sizes, which confirms the findings of the original authors. In addition, it is noteworthy that both algorithms are rather easily implementable.

The two IG methods IG_RS and IG_RS$_{LS}$ show, when compared to these other methods, an excellent performance. Consider first IG_RS, which does not include a local search phase, that is, it uses only construction heuristics. This algorithm is substantially more effective than other more complex alternatives like or GA_REEV or GA_RMA and provides better results than ILS. In fact, it even can compete with the much more difficult to implement M-MMAS and PACO algorithms; with these two algorithms it has in common that it relies strongly on constructive algorithms (recall that ants in ant algorithms are essentially randomized construction heuristics). However, differently from M-MMAS and PACO, IG_RS does not use additional local search; hence it is the best performing constructive algorithm for the PFSP currently known. Additionally, IG_RS is very easy to implement since essentially it is only a minor variation of NEHT. If local search is added, as done in IG_RS$_{LS}$, the performance of the iterated greedy algorithm further improves significantly. In fact, IG_RS$_{LS}$ gives the lowest $\overline{ARPD}$ value across all the algorithms in the comparison, surpassing the much more complex HGA_RMA by a noticeable margin in all instance sizes. Furthermore, IG_RS$_{LS}$ was able to find the optimum solution in all repetitions for the 10 instances of dimension 50×5 under the considered stopping criterion.

To validate the statistical significance of the observed differences in solution quality, we performed an analysis of variance. This analysis has a single factor which is the type of algorithm with 14 levels. The response variable is given by the RPD of every instance. (We consider all $R = 5$ repetitions in the experiment so to increase the power of the analysis.) In Figure 6 we give the means and confidence intervals for the various algorithms.

`[Insert Figure 6 about here]`

As can be seen, many of the differences are statistically significant. IG_RS$_{LS}$ is statistically better than IG_RS and all the other algorithms with the only exception being HGA_RMA. A more careful examination of the experimental data shows that IG_RS$_{LS}$ yields better results than HGA_RMA in 80 out of 120 instances, both algorithms obtain the same solution quality for 32 instances and for only 8 instances HGA_RMA gives slightly better results than IG_RS$_{LS}$. With these data it is expected that one would eventually find statistically significant differences among both methods when increasing the number of runs, on which the statistics are based.

The aforementioned results depict the "overall" performance of the tested algorithms. However, many of the instances in Taillard's set are very easy as can be seen by the fact that at the time of the writing of this paper there are only 31 open instances in the benchmark set of which almost all (28) belong to the instances of size 50×20, 100×20 or 200×20. On a selection of these hardest instances, we run a detailed comparison of the three best performing algorithms, IG_RS$_{LS}$, HGA_RMA, and PACO. In particular, each algorithm was run for 100 independent trials on each of the instances `ta051`, `ta081`, `ta101` and we measured the cumulative empirical distribution of the percentage deviation from the best known solution returned by the algorithms when stopping them after $n \cdot (m/2) \cdot 60$ milliseconds on the same PC as before. The plots of the respective distributions can be found in Figure 7. As can be seen from these plots, IG_RS$_{LS}$'s solution quality distribution is in all cases to the left of the other algorithms' ones. This indicates that for any fixed bound on the solution quality, IG_RS$_{LS}$ has a much higher probability of reaching it than its competitors. Hence, we can conclude that for these instances, IG_RS$_{LS}$ performs much better than its competitors. To test the statistical significance of these differences, we run for each instance pairwise comparisons among the algorithms using the Mann-Whitney test and the adjustment method of Holm because of multiple comparisons. The result was that all the pairwise differences were statistically significant at the $\alpha = 0.05$ level (except for `ta051` all p-values were smaller than $2 \cdot 10^{-16}$; for `ta051` the p-values were always smaller than $8.1 \cdot 10^{-11}$).

`[Insert Figure 7 about here]`

The extensive experimental comparison of known algorithms for the PFSP does not include some of the most effective metaheuristics from the literature like the TSAB algorithm of Nowicki and Smutnicki (1996) the RY genetic algorithm with path relinking by Reeves and Yamada (1998) and the recent TSGW of Grabowski and Wodecki (2004). There are several reasons for this. These algorithms are very complex and a re-implementation takes very long coding times.

For example, TSAB includes several complex speed-ups in all parts of the algorithm, from the critical path concept to how the neighborhood is evaluated in an efficient way. Despite this fact we tried an independent re-implementation of some of these algorithms, in particular of TSAB and RY. However, these re-implementations yielded inferior results to those published, probably because several intricate details of the algorithms are not clear from the descriptions of the algorithms.* We did not try a re-implementation of TSGW, since it was only very recently published. (Note that the results we obtained with the re-implementations of the algorithms used in the experimental comparison of Table 3 match rather well the results given in the original papers.)

Given these problems, we decided to compare the peak performance of IG_RS$_{LS}$ to the published results for TSAB, RY, and TSGW. In particular, we compare the best C_{max} obtained by our IG_RS$_{LS}$ after 5 independent runs of between 50 and 360 minutes of CPU time on our PC with an Athlon XP 1600+ processor (1400 MHz) with the best results of TSAB (time unknown), the results of RY (between approximately 48 and 188 minutes on a DEC Alpha 600 5/266 mainframe) and the best results of TSGW (between 1.3 and 4.5 minutes on a IBM RS6000/43P/200 mainframe). Here, only the hardest instances of Taillard, i.e. sizes 50×20, 100×20 and 200×20 are considered.† The results for C_{max} values and RPD, along with the lower and upper bounds from Taillard (2004) are given in Table 4.

`[Insert Table 4 about here]`

The results indicate that despite the simplicity of IG_RS$_{LS}$, the performance is very competitive from a solution quality point of view. For the 50×20 instances, IG_RS$_{LS}$ obtained better results than the other three algorithms and it even managed to improve the best known solutions for three instances (these are marked with $*$). The new best solutions are given in the appendix. However, the results for IG_RS$_{LS}$ deteriorate with instance size. In the 100×20 group, IG_RS$_{LS}$ is overall still the best performing algorithm for the majority of the instances; for the 200×20 instances the results are inferior to those of TSAB and TSGW, although they are still signifi-

*We contacted the original authors in each case. Unfortunately, in one case we did not receive any answer at all although we sent the requests several times; in the second case the answers were not fully sufficient to match the results of the published articles. In neither case we obtained the source code.

†Note that we are aware of the fact that our computation times are significantly larger than those given in these other three papers. However, our implementation is also a very basic one that does not use any intricate, problem specific speed-ups. The main intention of this comparison is to show that with respect to peak performance, our IG_RS$_{LS}$ can achieve results comparable to those of the best performing methods for the PFSP from literature.

cantly better than RY. It is important to remark that many of the speed-ups and tweaks used by TSAB, RY, or TSGW could be added to the local search of IG_RS$_{LS}$ and most probably the computation times to reach the same solution quality with IG_RS$_{LS}$ would drop strongly. However this would defeat the main purpose of the proposed IG_RS$_{LS}$ method, namely to develop a simple but at the same time very effective algorithm.

5 Conclusions

In this paper we have proposed an Iterated Greedy (IG) algorithm based on the NEH constructive heuristic of Nawaz et al. (1983) for the permutation flowshop problem with the Makespan criterion. The IG method makes use of a destruction operator that randomly removes some jobs from the sequence and a construction operator that reinserts the previously removed jobs following the NEH heuristic. This basic version of IG has been enhanced by considering an additional local search phase, which led to further significant improvements.[‡]

We have used experimental design analysis to tune the only two parameters of the IG algorithm. The tuned IG has been tested against a set of other 12 well performing algorithms from the PFSP literature, including some very recent ones. The results show that both versions of the proposed IG method (with and without local search) perform very well, especially when considering their conceptual simplicity. The IG method with local search resulted to be the most effective method in the experimental comparison. We have also compared this latter algorithm with the published results of three of the most effective local search algorithms from the literature. This comparison showed that the peak performance of our simple IG algorithm is comparable or sometimes better to those of much more complex methods. In addition, our algorithm was able to improve the best known solutions for three hard PFSP instances from the benchmark set of Taillard (1993), which is a remarkable result given the high level of sophistication the state-of-the-art in PFSP solving has reached.

This was the first time that IG was applied to scheduling problems and the computational results show its great promise. In fact, the overall performance of the IG algorithm can be regarded as very good since it does not make use of problem specific knowledge like the critical paths concept or extensive speed-ups as used in other published methods. Therefore, extensions

[‡]All the code used in this paper, as well as the proposed algorithms, benchmarks and results are available upon request from the authors.

of IG to flowshop problems with other objectives or features like sequence-dependent setup times are straightforward. Since also in many scheduling problems different from flowshops constructive algorithms play a significant role, we can expect that even a much larger variety of scheduling problems will profit from the ideas underlying iterated greedy.

Acknowledgments

Rubén Ruiz is partly funded by the Polytechnic University of Valencia, Spain, under an interdisciplinary project, and by the Spanish Department of Science and Technology (research project ref. DPI2001-2715-C02-01).

References

Aldowaisan, T. and Allahvedi, A. (2003). New heuristics for no-wait flowshops to minimize makespan. *Computers & Operations Research*, 30:1219–1231.

Beasley, J. E. (2004). OR-Library. `http://mscmga.ms.ic.ac.uk/info.html`.

Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B630–B637.

Chandrasekharan, R. and Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426–438.

Chen, C.-L., Vempati, V. S., and Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80(2):389–396.

Dannenbring, D. G. (1977). An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182.

Framinan, J. M., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Grabowski, J. and Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, 31(11):1891–1909.

Jacobs, L. W. and Brusco, M. J. (1995). A local search heuristic for large set-covering problems. *Naval Research Logistics Quarterly*, 42(7):1129–1140.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.

Lourenço, H. R., Martin, O., and Stützle, T. (2002). Iterated local search. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell, MA.

Marchiori, E. and Steenbeek, A. (2000). An evolutionary algorithm for large set covering problems with applications to airline crew scheduling. In Cagnoni, S. et al., editors, *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000*, volume 1803 of *Lecture Notes in Computer Science*, pages 367–381. Springer Verlag, Berlin.

Moccellin, J. V. and dos Santos, M. O. (2000). An adaptive hybrid metaheuristic for permutation flowshop scheduling. *Control and Cybernetics*, 29(3):761–771.

Montgomery, D. C. (2000). *Design and Analysis of Experiments*. John Wiley & Sons, fifth edition.

Murata, T., Ishibuchi, H., and Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, 30(4):1061–1071.

Nawaz, M., Enscore, Jr, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175.

Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA, The International Journal of Management Science*, 17(6):551–557.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, New Jersey, second edition.

Reeves, C. and Yamada, T. (1998). Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6(1):45–60.

Reeves, C. R. (1993). Improving the efficiency of tabu search for machine scheduling problems. *Journal of the Operational Research Society*, 44(4):375–382.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13.

Rinnooy Kan, A. H. G. (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague, The Netherlands.

Ruiz, R. and Maroto, C. (2004a). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*. In press.

Ruiz, R. and Maroto, C. (2004b). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine elegibility. *European Journal of Operational Research*. Accepted for publication.

Ruiz, R., Maroto, C., and Alcaraz, J. (2004a). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*. In press.

Ruiz, R., Maroto, C., and Alcaraz, J. (2004b). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, the International Journal of Management Science*. Accepted for publication.

Stützle, T. (1998). Applying iterated local search to the permutation flow shop problem. Technical report, AIDA-98-04, FG Intellektik, TU Darmstadt.

Stützle, T. (1998). An ant approach to the flow shop problem. In *Proceedings of the Sixth European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag Mainz, Wissenschaftsverlag, Aachen, Germany.

Suliman, S. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64:143–152.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):67–74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Taillard, E. (2004). Summary of best known lower and upper bounds of Taillard's instances. `http://ina.eivd.ch/collaborateurs/etd/problemes.dir/ordonnancement.dir/%ordonnancement.html`.

Turner, S. and Booth, D. (1987). Comparison of heuristics for flow shop sequencing. *OMEGA, The International Journal of Management Science*, 15(1):75–78.

Wang, L. and Zheng, D. Z. (2003). An effective hybrid heuristic for flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 21:38–44.

Werner, F. (1993). On the heuristic solution of the permutation flow shop problem by path algorithms. *Computers & Operations Research*, 20(7):707–722.

Widmer, M. and Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41(2):186–193.

Wodecki, M. and Bożejko, W. (2002). Solving the flow shop problem by parallel simulated annealing. In Wyrzykowski, R., Dongarra, J., Paprzycki, M., and Waśniewski, J., editors, *Parallel Processing and Applied Mathematics, 4th International Conference, PPAM 2001*, volume 2328 of *Lecture Notes in Computer Science*, pages 236–244. Springer-Verlag.

A Permutations of the new best solutions for Taillard benchmark

New best solution for `ta054`:
$n = 50$, $m = 20$, $C_{max} = 3723$
permutation = 5 11 14 21 30 13 24 12 7 45 35 20 19 31 25 37 3 44 33 17 43 46 48 29 23 49 40 39 32 26 47 50 9 42 22 6 38 10 15 36 4 27 2 18 8 1 16 41 34 28

New best solution for `ta056`:
$n = 50$, $m = 20$, $C_{max} = 3681$
permutation = 14 37 3 5 18 13 33 20 8 21 42 49 50 40 43 28 19 32 46 30 6 45 4 39 36 47 24 22 1 2 44 31 17 25 10 16 11 26 15 48 7 41 23 27 29 34 9 35 38 12

New best solution for `ta060`:
$n = 50$, $m = 20$, $C_{max} = 3756$
permutation = 33 12 19 8 22 14 2 50 9 40 1 11 3 36 34 32 25 47 16 29 20 35 31 27 18 42 10 37 44 23 28 5 17 38 13 45 41 21 15 7 24 39 6 26 49 46 43 30 48 4

machines (i)	jobs (j)							
	1	2	3	4	5	6	7	8
1	456	654	852	145	632	425	214	654
2	789	123	369	678	581	396	123	789
3	654	123	632	965	475	325	456	654
4	321	456	581	421	32	147	789	123
5	456	789	472	365	536	852	654	123
6	789	654	586	824	325	12	321	456
7	654	321	320	758	863	452	456	789
8	789	147	120	639	21	863	789	654

Table 1: Processing times p_{ij} for instance `Car8`.

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:n	34.4078	16	2.15049	478.96	0.0000
B:m	51.0534	3	17.0178	3790.24	0.0000
C:Destruction	1.01521	6	0.169202	37.69	0.0000
D:Temperature	0.0256892	5	0.00513783	1.14	0.3346
INTERACTIONS					
AB	21.2242	48	0.44217	98.48	0.0000
AC	2.04562	96	0.0213086	4.75	0.0000
AD	0.385624	80	0.0048203	1.07	0.3096
BC	0.886839	18	0.0492688	10.97	0.0000
BD	0.0848294	15	0.00565529	1.26	0.2196
CD	0.118968	30	0.00396559	0.88	0.6492
RESIDUAL	11.3954	2538	0.0044899		
TOTAL (CORRECTED)	122.643	2855			

Table 2: ANOVA table for the experiment on tuning the parameters of IG.

Instance	NEHT	GA_RMA	HGA_RMA	SA_OP	SPIRIT	GA_CHEN	GA_REEV	GA_MIT	ILS	GA_AA	M-MMAS	PACO	IG_RS	IG_RS$_{LS}$
20×5	3.35	0.26	0.04	1.09	4.33	4.15	0.62	0.80	0.49	0.94	0.04	0.21	0.04	0.04
20×10	5.02	0.73	0.13	2.63	6.07	5.18	2.04	2.14	0.59	1.54	0.15	0.37	0.25	0.06
20×20	3.73	0.43	0.09	2.38	4.44	4.26	1.32	1.75	0.36	1.43	0.06	0.24	0.21	0.03
50×5	0.84	0.07	0.02	0.52	2.19	2.03	0.21	0.30	0.20	0.36	0.03	0.01	0.04	0.00
50×10	5.12	1.71	0.72	3.51	6.04	6.54	2.06	3.55	1.48	3.72	1.40	0.85	1.06	0.56
50×20	6.26	2.74	1.28	4.52	7.63	7.74	3.56	5.09	2.20	4.69	2.18	1.59	1.82	0.94
100×5	0.46	0.07	0.02	0.30	1.06	1.35	0.17	0.27	0.18	0.32	0.04	0.03	0.05	0.01
100×10	2.13	0.62	0.29	1.48	3.01	3.80	0.85	1.63	0.68	1.72	0.47	0.27	0.39	0.20
100×20	5.23	2.75	1.66	4.63	6.74	8.15	3.41	4.87	2.55	4.91	2.59	2.09	2.04	1.30
200×10	1.43	0.43	0.20	1.01	2.07	2.76	0.55	1.14	0.56	1.27	0.23	0.27	0.34	0.12
200×20	4.41	2.31	1.48	3.81	4.97	7.24	2.84	4.18	2.24	4.21	2.26	1.92	1.99	1.26
500×20	2.24	1.40	0.96	2.52	12.58	4.72	1.66	3.34	1.25	2.23	1.15	1.09	1.13	0.78
Average	3.35	1.13	0.57	2.37	5.09	4.83	1.61	2.42	1.06	2.28	0.88	0.75	0.78	0.44

Table 3: Average relative percentage deviation over the optimum solution value or best known upper bound for Taillard instances and for the methods evaluated with the termination criterion set at $n \cdot (m/2) \cdot 60$ milliseconds elapsed time.

Instance	IG_RS$_{LS}$		TSGW		RY		TSAB		LB - UB
	C_{max}	RPD	C_{max}	RPD	C_{max}	RPD	C_{max}	RPD	
$\mathbf{50 \times 20}$									
ta051	3852	0.05	3855	0.13	3861	0.29	3856	0.16	3771 - 3850
ta052	3705	0.03	3708	0.11	3709	0.13	3714	0.27	3668 - 3704
ta053	3645	0.11	3650	0.25	3651	0.27	3658	0.47	3591 - 3641
ta054	3723*	-0.03	3729	0.13	3726	0.05	3737	0.35	3635 - 3724
ta055	3614	0.08	3614	0.08	3614	0.08	3619	0.22	3553 - 3611
ta056	3681*	-0.11	3689	0.11	3690	0.14	3690	0.14	3667 - 3685
ta057	3705	0.00	3707	0.05	3711	0.16	3709	0.11	3672 - 3705
ta058	3694	0.08	3700	0.24	3699	0.22	3700	0.24	3627 - 3691
ta059	3753	0.27	3756	0.35	3760	0.45	3760	0.45	3645 - 3743
ta060	3756*	-0.29	3767	0.00	3767	0.00	3767	0.00	3696 - 3767
Average	3712.8	0.02	3717.5	0.15	3718.8	0.18	3721	0.24	
$\mathbf{100 \times 20}$									
ta081	6221	0.31	6223	0.34	6242	0.64	6238	0.58	6106 - 6202
ta082	6200	0.23	6210	0.39	6217	0.50	6210	0.39	6183 - 6186
ta083	6296	0.40	6283	0.19	6399	2.04	6296	0.40	6252 - 6271
ta084	6303	0.54	6278	0.14	6288	0.30	6278	0.14	6254 - 6269
ta085	6327	0.21	6331	0.27	6329	0.24	6351	0.59	6262 - 6314
ta086	6364	0.00	6382	0.28	6380	0.25	6406	0.66	6302 - 6364
ta087	6292	0.38	6287	0.30	6302	0.54	6298	0.48	6184 - 6268
ta088	6401	0.00	6425	0.37	6433	0.50	6423	0.34	6315 - 6401
ta089	6286	0.18	6286	0.18	6297	0.35	6291	0.25	6204 - 6275
ta090	6441	0.11	6441	0.11	6448	0.22	6441	0.11	6404 - 6434
Average	6313.1	0.23	6314.6	0.26	6333.5	0.56	6323.2	0.39	
$\mathbf{200 \times 20}$									
ta101	11248	0.47	11213	0.16	11272	0.69	11213	0.16	11152 - 11195
ta102	11252	0.44	11242	0.35	11299	0.86	11249	0.41	11143 - 11203
ta103	11391	0.53	11367	0.32	11410	0.70	11382	0.45	11281 - 11331
ta104	11343	0.43	11295	0.01	11347	0.47	11309	0.13	11275 - 11294
ta105	11287	0.25	11267	0.07	11290	0.28	11265	0.05	11259 - 11259
ta106	11257	0.61	11212	0.21	11250	0.55	11212	0.21	11176 - 11189
ta107	11401	0.29	11389	0.18	11438	0.62	11387	0.17	11337 - 11368
ta108	11363	0.26	11354	0.18	11395	0.54	11362	0.25	11301 - 11334
ta109	11230	0.34	11220	0.25	11263	0.63	11241	0.44	11145 - 11192
ta110	11336	0.22	11313	0.02	11335	0.21	11313	0.02	11284 - 11311
Average	11310.8	0.38	11287.2	0.17	11329.9	0.55	11293.3	0.23	

Table 4: Best C_{max} and Relative Percentage Deviation (RPD) found by the proposed IG_RS$_{LS}$ algorithm and some of the state-of-the-art methods from the literature. LB-UB gives the best known lower and upper bounds as of April 2004.

procedure *LocalSearch_Insertion* (π)
    improve $:=$ true;
    while $($*improve* $=$ true$)$ **do**
        improve $:=$ false;
        for $i := 1$ **to** n **do**
            remove a job k at random from π (without repetition)
            $\pi' :=$ best permutation obtained by inserting k in any possible positions of π;
            if $C_{max}(\pi') < C_{max}(\pi)$ **then**
                $\pi := \pi'$;
                improve $:=$ true;
            endif
        endfor
    endwhile
    return π
end

Figure 1: Local search procedure in the insertion neighborhood

```
procedure IteratedGreedy_for_PFSP
    π := NEH_heuristic;
    π := LocalSearch_Insertion(π);
    π_b := π;
    while (termination criterion not satisfied) do
        π' := π;                              % Destruction phase
        for i := 1 to d do
            π' := remove one job at random from π' and insert it in π'_R;
        endfor
        for i := 1 to d do                    % Construction phase
            π' := best permutation obtained by inserting job π_R(i) in all possible positions of π';
        endfor
        π'' := LocalSearch_Insertion(π');     % Local Search
        if C_max(π'') < C_max(π) then         % Acceptance Criterion
            π := π'';
            if C_max(π) < C_max(π_b) then     % check if new best permutation
                π_b := π;
            endif
        elseif (random ≤ exp{−(C_max(π'') − C_max(π))/Temperature}) then
            π := π'';
        endif
    endwhile
    return π_b
end
```

Figure 2: Iterated Greedy (IG) algorithm with local search phase. $random$ is a random number distributed uniformly in $[0, 1]$.

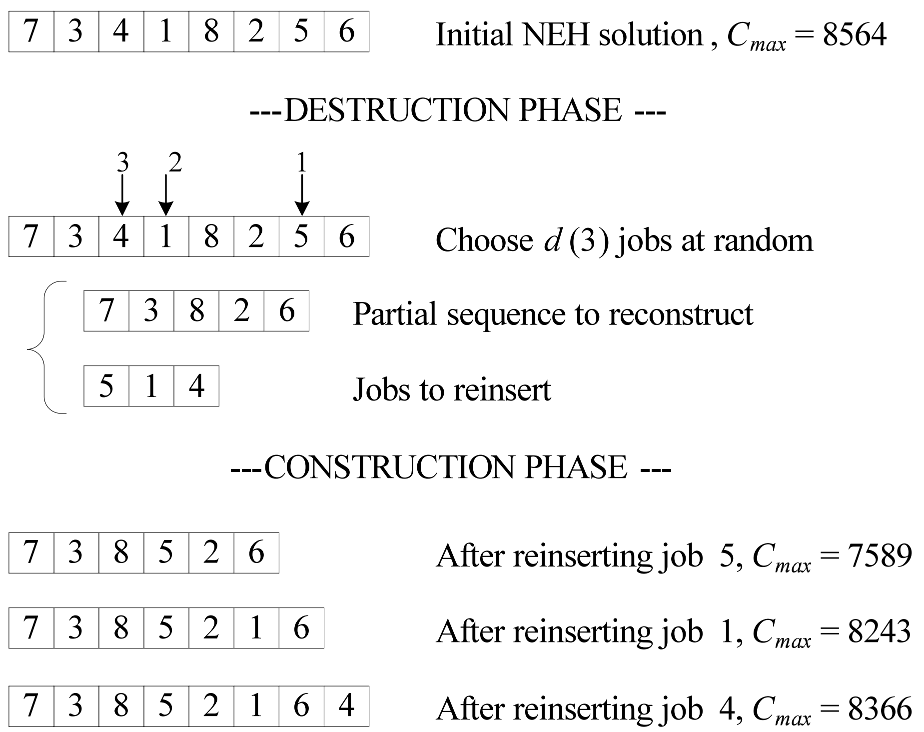


Figure 3: Example for the application of one iteration of the IG algorithm (without local search) to instance `Car8`.

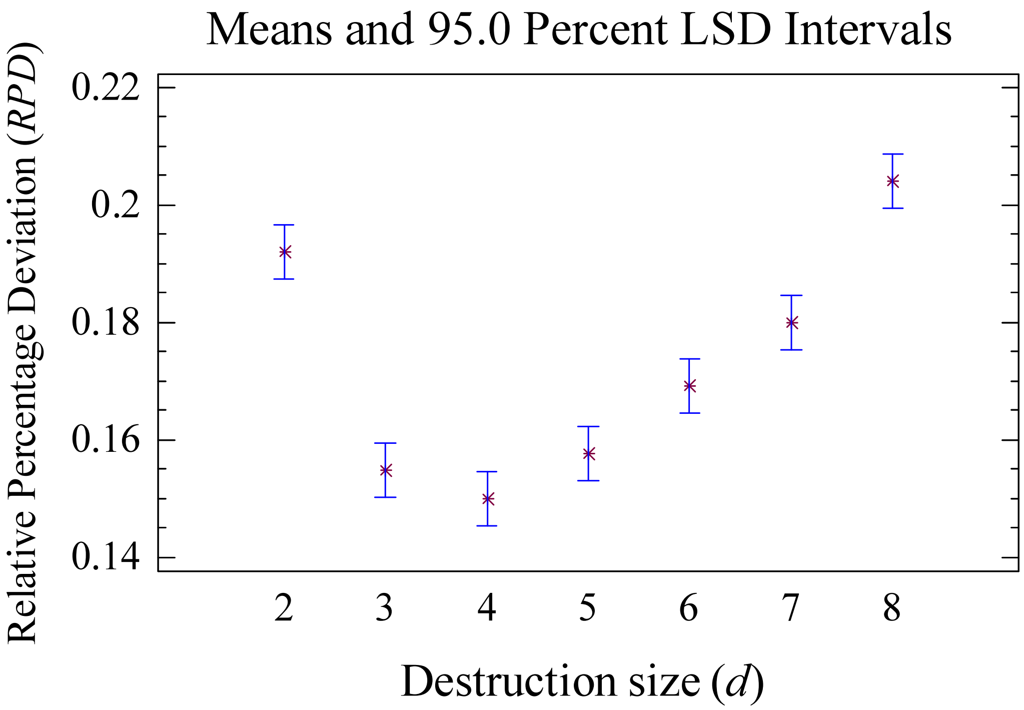


Figure 4: Plot of the average percentage deviation from best known solutions and the corresponding confidence intervals for various settings of the destruction size.

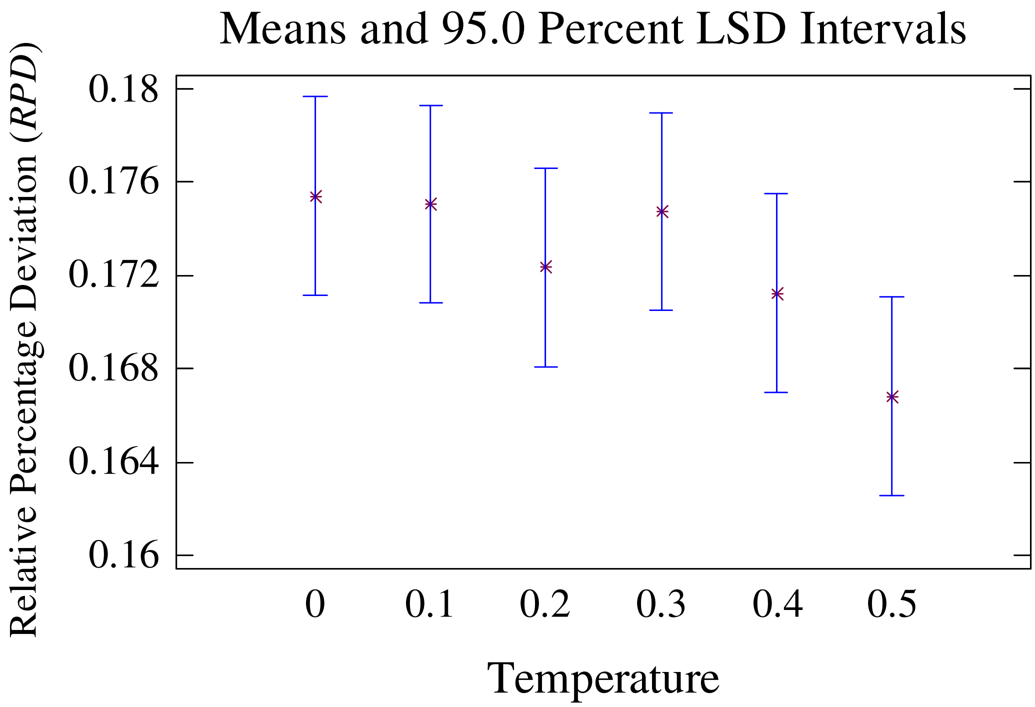


Figure 5: Plot of the average percentage deviation and the corresponding confidence intervals for various settings of the parameter T, which determines the temperature used in the acceptance criterion.

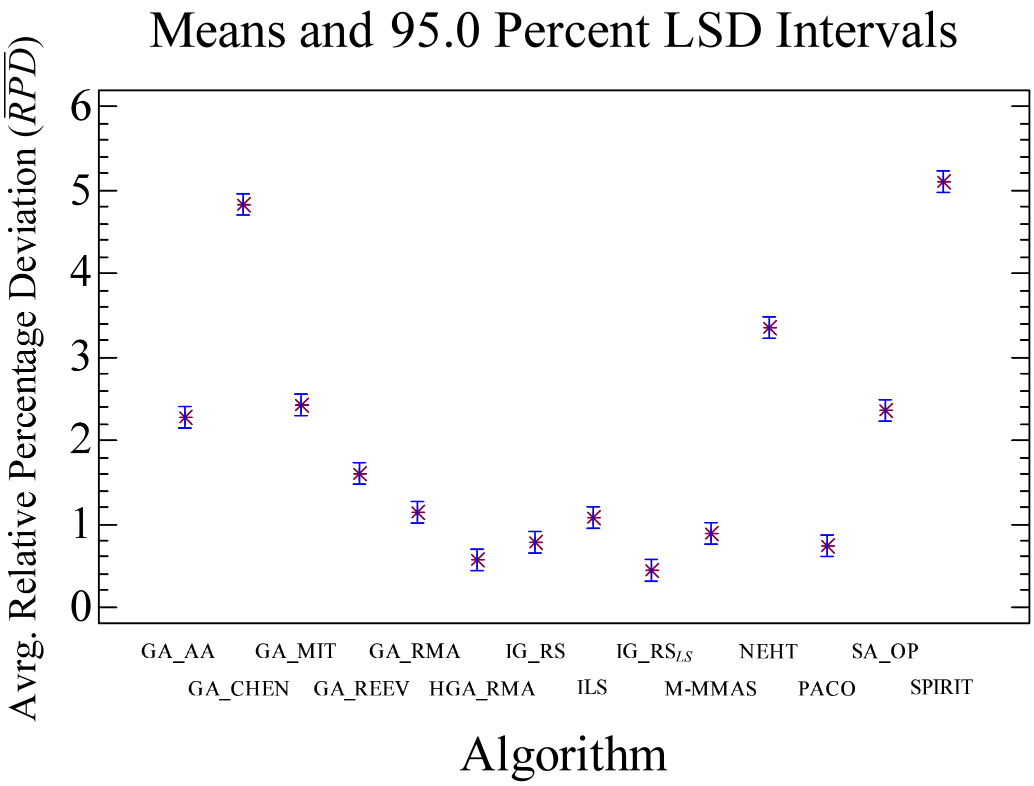
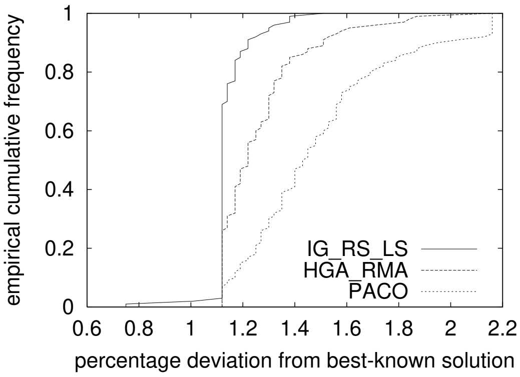
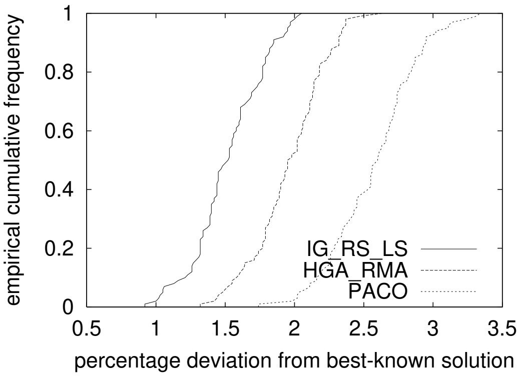


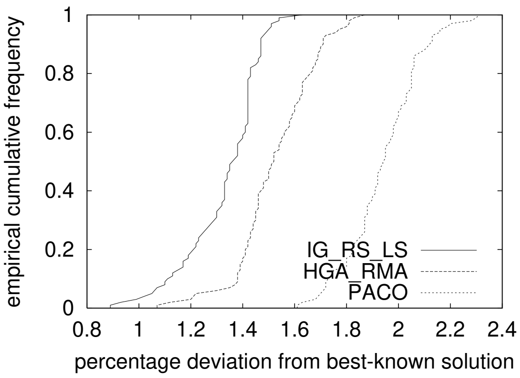
Figure 6: Plot of the average percentage deviation from best known solutions and the corresponding confidence intervals for the algorithms tested.



(a) Instance ta051



(b) Instance ta081



(c) Instance ta101

Figure 7: Plots of the empirical cumulative solution quality distribution for the three best performing algorithms from our analysis (IG_RS$_{LS}$, HGA_RMA, and PACO) for three instances of size 50×20 (`ta051`), 100×20 (`ta081`), and 200×20 (`ta101`).